APPLICATION UNDER UNITED STATES PATENT LAWS

Atty. Dkt. No. PW 249727
(M#)

Invention: INGRESS THROTTLING VIA ADAPTIVE INTERRUPT DELAY SCHEDULING

Inventor (s): Patrick L. CONNOR
Daniel R. GAUR
Eric K. MANN
Gary Y. TSAO
Michael C. GIBSON

00909
Pillsbury Winthrop LLP

This is a:

☐ Provisional Application

☒ Regular Utility Application

☐ Continuing Application
☐ The contents of the parent are incorporated by reference

☐ PCT National Phase Application

☐ Design Application

☐ Reissue Application

☐ Plant Application

☐ Substitute Specification
Sub. Spec Filed ______
in App. No. ______

☐ Marked up Specification re
Sub. Spec. filed ______
In App. No / ______

SPECIFICATION

Intel Ref: : P12819
Pillsbury Ref: 81674/249727

INGRESS THROTTLING VIA ADAPTIVE INTERRUPT DELAY SCHEDULING

Reservation of Copyright

[0001] This patent document contains information subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent, as it appears in the U.S. Patent and Trademark Office files or records but otherwise reserves all copyright rights whatsoever.

BACKGROUND

[0002] Aspects of the present invention relate to communications. Other aspects of the present invention relate to packet receive interruption.

[0003] Physical devices in a computer system frequently use an "interrupt" mechanism to notify the occurrence of certain events. For instance, an I/O controller might generate an interrupt upon successfully transmitting a packet or upon receiving an incoming packet. Most I/O controllers, such as Ethernet Media Access Controllers (MACs), use interrupts as a means of notifying the arrival of incoming packets. An I/O controller may be capable of receiving tens or hundred of thousands of packets per second. Typically, an I/O controller generates a "receive interrupt" after a new packet is received from the network. The interrupt may trigger an "interrupt handler" to process the newly-arrived packet(s). The interrupt handler may then verify the cause of the interrupt and then perform necessary post-interrupt operations.

[0004] Modern processors are usually optimized for processing streams of data, such as the data sent and received over a network connection. To better process data streams, processors internally overlap arithmetic and memory operations. To implement this overlapped execution, the processor has a data processing 'pipeline' comprising a plurality of pipeline stages. Interrupts force the processor to stop, cancel and drain its internal pipeline, thereby disrupting existing processing. Frequent disruptions may reach such a level that the processor can process only a small portion (if any) of received data. In effect, system data throughput can be drastically reduced.

[0005] To improve 'system throughput', a system may incorporate faster I/O devices. However, faster I/O devices can create even more interrupts. Alternatively, a system may be made to process data more efficiently by adding more stages to the processor pipeline. Unfortunately, neither of these approaches directly addresses the problem of tuning the interaction between the I/O controller and the processor.

[0006] The pathology of a system that spends majority of its time processing receive interrupts, which are subsequently dropped, is referred to as livelock. A primary cause of livelock is often an interrupt storm referring to a rapid succession of interrupts that preempts other tasks. Under heavy sustained network loads, interrupt storms can occur, leaving upper layers in a processing pipeline starved for CPU cycles. When such starved layers are no longer able to buffer packets, they subsequently drop the packets. Therefore, even though a high level of ingress throughput may be observed at the outset, the final consumer may see little or no throughput at all.

[0007] To alleviate this problem, some high-speed I/O controllers implement a method of receive (ingress) interrupt moderation to improve the efficiency of interrupts

asserted to the processor. This allows for a single interrupt to signal more than one received packet based on the load situation at the lowest layer (network and/or device driver). Fig. 1 illustrates a framework 100 that employs the solution. I/O controller 110 sends receive interrupts to a host 140 via a bus 130. To avoid interrupt storms, the I/O controller employs a load-based interrupting mechanism 120 that moderates the receive interrupt according to the load situation. Upon receiving ingress interrupts, the protocol stack 150 in the host 140 processes the received packets.

[0008] With this solution, under a heavy load situation, packets can be possibly dropped at any layer of the protocol stack 150. Particularly, when a higher layer drops packets, valuable resources may be wasted. For example, if the host 140 performs all necessary processing to deliver a packet to a higher layer of the protocol stack 150, such as TCP/IP, and then such packets are ultimately dropped, multiple system resources are wasted, including the bus bandwidth used to transfer the packet, the CPU cycles associated with handling the receive interrupts, and any operations, such as checksum verification and decryption, performed by the layers prior to the layer where the packets are dropped.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] The present invention is further described in terms of exemplary embodiments, which will be described in detail with reference to the drawings. These embodiments are non-limiting exemplary embodiments, in which like reference numerals represent similar parts throughout the several views of the drawings, and wherein:

[0010] Fig. 1 (prior art) illustrates a framework, in which each receive interrupt sent from an I/O controller representing a plurality of packets is determined based on packet load;

[0011] Fig. 2 depicts a framework, in which a receive interrupt from an I/O controller to a host is sent with a delay computed based on backlog situation of the host, according to embodiments of the present invention;

[0012] Fig. 3 depicts internal structures of an I/O controller in relation to a host that provides backlog information for the I/O controller to determine an interrupt delay, according to embodiments of the present invention;

[0013] Fig. 4 is an exemplary flowchart of a process, in which an I/O controller and a host communicate about received packets via a receive interrupt with a delay that is adaptively adjusted based on the backlog situation of the host, according to embodiments of the present invention;

[0014] Fig. 5 is an exemplary flowchart of a process, in which an I/O controller adaptively determines a delay in sending a receive interrupt to a host based on the backlog situation of the host, according to embodiments of the present invention;

[0015] Fig. 6 is an exemplary flowchart of a process, in which a host processes packets upon intercepting a receive interrupt and returns processed packets;

[0016] Fig. 7 is an exemplary plot of a plurality of constant functions, each of which represents a delay function within a particular backlog zone, according to an embodiment of the present invention; and

[0017] Fig. 8 is an exemplary plot of a plurality of linear functions, each of which represents a delay function within a particular backlog zone, according to an embodiment of the present invention.

DETAILED DESCRIPTION

[0018] The processing described below may be performed by a properly programmed general-purpose computer alone or in connection with a special purpose computer. Such processing may be performed by a single platform or by a distributed processing platform. In addition, such processing and functionality can be implemented in the form of special purpose hardware or in the form of software or firmware being run by a general-purpose or network processor. Any data handled in such processing or created as a result of such processing can be stored in any memory as is conventional in the art. By way of example, such data may be stored in a temporary memory, such as in the RAM of a given computer system or subsystem. In addition, or in the alternative, such data may be stored in longer-term storage devices, for example, magnetic disks, rewritable optical disks, and so on. For purposes of the disclosure herein, a computer-readable media may comprise any form of data storage mechanism, including such existing memory technologies as well as hardware or circuit representations of such structures and of such data.

[0019] Fig. 2 depicts a framework 200, in which a receive interrupt 220 from an I/O controller 110 to a host 140 is sent with a delay computed based on backlog information 230 from the host, according to embodiments of the present invention. The framework 200 comprises an I/O controller 110, a host 140, and a bus 130 through which the I/O controller 110 and the host 140 send information to each other. The I/O controller 110 is responsible for receiving packets, for notifying the host 140 about received packets, and for sending the received packets to the host 140 for further processing. Upon being notified by the I/O controller 110, the host 140 processes the received packets sent from the I/O controller 110

and notifies the I/O controller 110 whenever the processing of a received packet is completed. To notify the host 140, the I/O controller 110 sends a receive interrupt 220 via the bus 130.

[0020] Upon receiving packets, the I/O controller 110 stores the received packets in buffers. It then, at an appropriate time, notifies the host 140 about the received packets by asserting receive interrupt 220. When the host 140 completes its processing of a packet, to return the packet to the I/O controller to signal its completion, it may not necessarily send the packet back to the I/O controller 110. Instead, it may simply notify the I/O controller 110 of which packet has been processed. Through this mechanism, the I/O controller 110 is aware of the progress of the host 140 with respect to processing the received packets. That is, the I/O controller 110 knows how many packets have been processed and how many packets that are still backlogged in the host 140, waiting to be processed.

[0021] The I/O controller 110 and the host 140 may operate asynchronously. The I/O controller 110 may keep receiving and buffering packets while the host 140 is processing the packets that are handed over previously. To determine when it is appropriate to indicate the packets so far received, the I/O controller 110 employs a backlog-based interrupting mechanism 210. The backlog-based interrupting mechanism 210 determines an appropriate time to notify the host 140 (about the received packets that are buffered) by asserting receive interrupt 220 to the host 140.

[0022] The appropriate time may be computed as the present time plus a delay. The backlog-based interrupting mechanism 210 computes such a delay according to the present backlog situation on the host 140. Present backlog situation may be assessed based on, for instance, the percentage of the packets that have been returned from the host 140. Such computed delay is therefore adaptive to the backlog situation. For example, if presently there

is no backlog (i.e., all or substantial number of the packets have been returned), the delay may be zero. If the host 140 is presently very backed up (e.g., a large percentage of the packets that are handed over previously are still not yet returned), the delay may be adaptively made longer. Based on this adaptive delay, the I/O controller 110 will not assert receive interrupt 220, for a period of time specified by the delay.

[0023] The host 140 includes a protocol stack 150 that may comprise a plurality of layers. Each layer of the protocol stack 150 processes relevant packets. Whenever a packet is processed, the protocol stack 150 notifies the I/O controller 110.

[0024] Fig. 3 depicts the internal structures of the I/O controller 110 in relation to the host 140 that provides backlog information 230 for the I/O controller 110 to determine an interrupt delay, according to embodiments of the present invention. The I/O controller 110 comprises a packet receiver 330, a buffer allocation mechanism 310, a packet population mechanism 320, a packet buffer 340, and the backlog-based interrupting mechanism 210. The buffer allocation mechanism 310 is responsible for allocating packet buffers that are used for storing received packets. The packer receiver 330 is responsible for receiving packets that are transferred to the I/O controller 110. Upon receiving such packets, the packet receiver 330 invoke the packet population mechanism 320 to populate the received packets in the packet buffer 340.

[0025] The backlog based interrupting mechanism 210 includes a delay determination mechanism 360 and an interrupt generation mechanism 350. The delay determination mechanism 360 gathers information related to the backlog situation of the host 140 (e.g., the percentage of the packets that have not been returned) and computes a delay accordingly. The relationship between backlog situation assessment and the computed delay may depend on

application needs and may be captured using some functions. Detailed discussion related to the computation of a delay is presented later in referring to Fig. 7 and Fig. 8.

[0026] The interrupt generation mechanism 350 uses the computed delay to control when to generate next receive interrupt that notifies the host 140 about received packets. That is, the computed delay is enforced via the interrupt generation mechanism 350 by not generating next interrupt until the delay is satisfied. In such scenarios, the delay may serve as a timer.

[0027] The host 140 includes an interrupt handler 370 and a protocol stack 150, which may comprise a plurality of layers of processing mechanisms 390 and a packet return mechanism 380. When the interrupt handler 370 intercepts a receive interrupt, it notifies the protocol stack 150. Different layers of the packet processing mechanism 390 in the protocol stack 150 may then selectively process the packets that are available. When the processing on a particular packet is completed, the packet return mechanism 380 notifies the I/O controller 110 about the completion.

[0028] Fig. 4 is an exemplary flowchart of a process, in which an I/O controller and a host communicate about received packets via a receive interrupt with a delay that is adaptively adjusted based on the backlog situation of the host, according to embodiments of the present invention. Packets are received first at 410 and then populated, at 420, into the packet buffer. The I/O controller 110 then determines when to issue an interrupt to inform the host 140 about the received packet. To do so, the I/O controller 110 computes, at 430, the interrupt delay based on the backlog situation at the host 140. Such determined delay is then asserted at 440. When the delay is satisfied, the I/O controller 110 generates, at 450, a receive interrupt and asserts the interrupt, at 460, to the host 140. Upon intercepting the receive

interrupt, the host 140 processes, at 480, the received packets and then returns, at 490, the processed packets to the I/O controller 110.

[0029] Fig. 5 is an exemplary flowchart of a process, in which the I/O controller 110 adaptively determines the delay based on the backlog situation of the host 140, according to embodiments of the present invention. A packet buffer is first allocated, at 510, for storing received packets. Packets are received at 520 and are populated in the packet buffer at 530. The backlog-based interrupting mechanism 210 then assesses, at 540, the backlog situation based on information related to returned packets. A delay is then accordingly determined at 550.

[0030] A receive interrupt will not be generated until the delay is satisfied. When the delay is not satisfied, determined at 560, the I/O controller 110 may keep receiving more packets and subsequently populating them in the packet buffer. When the delay is satisfied, the I/O controller 110 generates, at 570, a receive interrupt and sends, at 580, the interrupt to the host 140. The I/O controller 110 then sends, at 590, the received packets to the host 140.

[0031] Fig. 6 is an exemplary flowchart of a process, in which the host 140 processes packets upon intercepting a receive interrupt and returns a processed packet to the I/O controller when the processing is completed. A receive interrupt is intercepted first at 610. The interrupt signal is then processed at 620. Being notified that there are more received packets, the host 140 receives, at 630, the packets. Various layers of the packet processing mechanism 390 in the protocol stack 150 then proceeds to process, at 640, the packets in the buffer. For each packet that is processed, the packet return mechanism 380 returns, at 650, the processed packet to the I/O controller 110. The process continues until all the packets have been processed, determined at 660.

[0032] As described earlier, a delay in the context of the present invention may be computed based on current backlog situation, which may be assessed according to, for example, the percentage of packets that have been returned from the host 140. Backlog may be classified into a plurality of zones, each of which may correspond to a different level of backlog severity. For example, zone 1 may correspond to the situation that there is no backlog or small degree of backlog. Zone 2 may correspond to a medium degree of backlog and zone 3 may correspond to a severe backlog situation. Number of such zones employed may depend on application needs. When a fewer number of zones are used, the computation required to compute the delay may be reduced. When a larger number of zones are used, the backlog-based interrupting mechanism 210 may be tuned at a finer resolution.

[0033] The backlog-based interrupting mechanism 210 may compute a delay with respect to zone classification. When backlog information is used to determine a delay, the severity of current backlog situation affects the amount of delay. For example, when there is no backlog, no delay is needed. When there is a severe backlog, a large delay is necessary. Therefore, for each zone, a different computation may be applied to accordingly determine the delay necessary for that zone. Fig, 7 and 8 illustrate exemplary computation schemes to compute a delay based on backlog zone classification.

[0034] Fig. 7 is an exemplary plot of a plurality of constant functions, each of which represents a constant function used to compute a delay within a particular backlog zone, according to an embodiment of the present invention. In Fig. 7, the horizontal axis represents the severity of backlog and the vertical axis represents the amount of delay. There are three exemplary backlog zones illustrated, zone 1 (710), zone 2 (720), and zone 3 (730), corresponding to no backlog, some backlog, and severe backlog. Each zone may be defined

according to certain percentage of packets that have not been returned. For example, zone 1 710 may be defined as having less than 20% of packets that have not been returned. That is, more than 80% of the packets have been returned. Similarly, zone 2 (720) may be defined as having more than 50% of the packets returned. The severe backlog zone (zone 3 730) may be defined as having only less than 50% of the packets returned.

[0035] In Fig. 7, a plurality of constant functions are depicted and they are used to compute the delay with respect to each zone. For example, a delay level 2 (750) (corresponding to a constant) is used for zone 2 (720). That is, if a backlog situation is classified as zone 2 (i.e., more than 50% but less than 80% of packets have been returned), the resultant delay is specified by the delay level 2 (750). In the exemplary illustration in Fig. 7, the delay level 2 corresponds to “5 packet times”, meaning a delay of next 5 packets (or alternatively, do not send a receive interrupt for the next 5 received packets). Similarly, if a backlog situation is classified as zone 3 (i.e., more than 50% of the packets are not yet returned), the resultant delay is defined by a different constant function with delay level 3 (760), corresponding to a delay of “15 packet times”.

[0036] Different functions, other than constant functions, may also be employed to map a backlog severity level to a delay value. For example, a linear function may be employed. While a constant function provides a single value within each zone and two constant functions across two zones may introduce a sharp jump (e.g., the difference between the delay level 2 and the delay level 3 in Fig. 6), a linear function may provide a continuous mapping between the severity of backlog and the delay value. In addition, it is possible to define linear functions in such a way that the transition between adjacent zones is smooth.

[0037] Fig. 8 is an exemplary plot of a plurality of linear functions, each of which represents a delay function within a particular backlog zone, according to an embodiment of the present invention. There are three linear functions illustrated (delay function 1 810, delay function 2 820, and delay function 3 830) that define mappings between backlog severity and delay values across three backlog zones (zone 1 710, zone 2 720, and zone 3 730). Each linear function maps a backlog severity to a delay value spanning from a lower bound delay value to an upper bound delay value. For example, the delay function 1 810 defines the mapping between a backlog severity value in zone 1 (710) and a delay value between no delay (or 0 delay) to a delay level of "3 packet times". Similarly, the delay function 2 820 corresponds to a mapping, ranging from delay level "3 packet times" to "15 packet times", that describes the relationship between a backlog severity level within zone 2 and a delay value between lower bound "3 packet times" and upper bound "15 packet times".

[0038] Within each zone, a linear delay function proportionally maps a particular backlog to a delay level. That is, each different backlog severity will result in a different delay value. This is different from a constant mapping, where all backlog severity values within a same zone will result in a same delay value. Therefore, a linear delay function provides finer level of adaptivity.

[0039] In the illustrated example shown in Fig. 8, adjacent linear functions may be so designed that the transition between two adjacent zones is smooth. For example, since the upper bound of the delay function 2 820 ("15 packet times") is the same as the lower bound of the delay function 3 830, the transition between zone 2 720 and zone 3 730 will be smooth.

[0040] In a particular implementation, constant functions and linear functions may be mixed across different backlog zones. In addition, non-linear functions may also be

employed, either alone or together with constant or linear functions, to define the mapping between backlog severity level classifications and interrupt delays.

[0041] While the invention has been described with reference to the certain illustrated embodiments, the words that have been used herein are words of description, rather than words of limitation. Changes may be made, within the purview of the appended claims, without departing from the scope and spirit of the invention in its aspects. Although the invention has been described herein with reference to particular structures, acts, and materials, the invention is not to be limited to the particulars disclosed, but rather can be embodied in a wide variety of forms, some of which may be quite different from those of the disclosed embodiments, and extends to all equivalent structures, acts, and, materials, such as are within the scope of the appended claims.